technoculture:
an online journal
of technology in society

Published on *Technoculture: An Online Journal of Technology in Society* (http://tcjournal.org/drupal)

Critical Essay—One Hundred Thousand Billion Processes: Oulipian Computation and the Composition of Digital Cybertexts

Kevin Brock, North Carolina State University

Abstract:

Scholars of, or interested in, rhetoric have an opportunity to build upon the emerging body of work from the fields of software studies and critical code studies in order to explore the potential for meaning-making made possible through code and its expression(s). Over the last decade, rhetoric has significantly expanded to incorporate image, sound, video, and game play into its domain, especially in regards to rhetorical acts facilitated by computers. However, there has been relatively little scrutiny of the rhetorical value and agency of the procedural structures on which these acts are constructed.

In order to draw attention to how code works rhetorically, this article examines three Oulipian "cybertexts," works that a) are more interested in the "potential" texts they can create than the importance of any particular outcomes, and b) demonstrate their underlying mechanisms as integral components of their expression. There are several key observations for rhetoricians to be gained from these examinations, and the most notable is the capacity for *action* made possible through their composition in code and through their expressive performances. Each cybertext conveys meaning through its potential to induce change in human and technological audiences through the code and natural languages that comprise it.

Introduction

Scholars interested in rhetoric, writing, and communication have before them a fascinating and significant moment: accompanying the increasingly ubiquitous digital technologies that facilitate our day-to-day lives are an increasing variety of open tools for using those technologies in order to create novel *ways* of meaning-making. While the prevalent approaches to critical inquiry surrounding digital technologies—at least for studies of communicative interaction—focus on the end-user products of software mediation, we can and *should* turn our attention to the processes and procedures underlying these meaningful texts. As Noah Wardrip-Fruin notes, “[t]rying to interpret a work of digital media by looking only at the output is like interpreting a model solar system by looking only at the planets” (158). Instead, we must examine both the components of a system *and* how those components work together systematically to create meaningful results and activities. It is the intersection of compositional process and computational procedure, from Boolean logic to high-level programming language readability, where rhetors are enabled and constrained through digital contexts to influence their audiences so as to pursue particular forms of action.

Several literary movements have explored the possibilities of composing artistic works via computational structures. A mid-twentieth group of mathematicians and writers who called themselves the Oulipo (an acronym for ***Ou****vroir de* ***Li****ttérature* ***Po****tentielle*, or "workshop for potential literature") focused on literature generated within the constraints of algorithmic procedures. While their efforts produced works, including early hypertext fictions, that have divided critics for their emphasis on structure over content, the Oulipo nonetheless demonstrated that meaning-making could exist far beyond the bounds of conventional forms of expressive communication. Similarly, a number of poet-programmers in the late twentieth and early twenty-first century, including Mez, Jodi, and Antiorp, played with computer code languages and related symbols to create artistic pieces they called *codework* (via Alan Sondheim) as a way to push against the limits of natural language discourse.

Rhetoricians have spent the last several decades incorporating into their fold a wide range of communicative modes, means, and forms, scrutinizing how, following Aristotle, "the available means of persuasion" (I.ii.1) may include the visual (Kress and van Leeuwen), aural (Halbritter), and even procedural (Bogost) qualities of a particular argument and how effectively, and to what ends, those qualities influence a rhetor's message as it is communicated to his or her audience. These critical inquiries have adapted to the explosion of digital technologies for multimodal and multimedia communication, but very little has been explored yet in regards to how the development of digital technologies, and how they work, could provide insight into the rhetorical processes that anticipate and restrict expression through those technologies. For example, HTTP as a protocol for transferring information, and HTML as a means of marking up text, serve functions beyond rendering the visual layout of web pages when viewed in a browser, but our focus has centered on those rendered documents.

Meanwhile, over the past decade a growing number of scholars across a number of disciplines have heeded Lev Manovich's 2001 call (in his seminal *The Language of New Media*) to study the cultural and critical significances and practices of software. These scholars have done so *in part* by engaging computer code files, syntax, and languages as texts to be read as works of literature and as meaningful representations or expressions of various sociocultural ideologies and values—an approach addressed most directly in Matthew Fuller's *Behind the Blip*; in Fuller's *Software Studies* collection; and in Wardrip-Fruin's *Expressive Processing* —although, as Wardrip-Fruin observes, “examining code and examining processes are not the same thing” since each pursues a different, albeit overlapping, set of goals (163). That said, among the most intriguing advances made by those involved in software studies has been their explicit focus on what code says *as well as* on how it functions (i.e. how its processes work): the mechanics of software are placed on an equal level with the expressive performances of interaction they enable. Mark C. Marino articulates this call eloquently in his essay “Critical Code Studies,” when he argues for the need for critics to understand how code transmits and imparts meaning to human audiences in addition to understanding how it completes computational tasks.

Unfortunately, very few in either area have attempted to straddle both of these trajectories of inquiry so as to weave them together into a cohesive investigation of the possibilities of code as a form of rhetorically meaningful composition. Given the sheer amount of software development currently taking place across the globe (and not just by professionals creating polished consumer-based products), there is an opportunity to examine the varied existing and emerging rhetorical approaches being assumed by coders of all stripes to generate code texts *as well as* to create executable programs running on that code. This range of perspectives could offer a wealth of insight into understanding how software-oriented rhetors anticipate their audiences' actions in responding to and interacting with their works.

In this essay, I seek to offer rhetoricians interested in digital technologies, and critics of software interested in the rhetorical capabilities thereof, one such potential connection of these trajectories. Each group may incorporate within their critical studies the numerate and literate qualities and forms of writing as performances, generations of, and influences upon all types of action—what happens when, as Fuller notes, “computation […] comes into combination with what lies outside of code” (“Introduction” 5). Such an understanding of how this combination works culturally and rhetorically can best be accomplished through an exploration of computer programming language code and the rhetoric of procedure involved in code practices.

In order to follow this emergent line of inquiry, I will examine three digital texts that demonstrate the generative qualities of action made possible through the construction and execution of computer codes as meaningful communication (alongside text, image, motion, etc.), each of which work in a *hypermediate* fashion (via Bolter and Grusin), where the medium of a text draws attention to itself as an integral and meaningful part of the reading experience. These potential ranges of action —graphical, temporal, semiotic, procedural—are both accessible to their audiences (before the iterative executions of their codes) and obscured from them (when the audience engages with the codes' expressions), resulting in writing and reading experiences that, following the experiments of the Oulipo, complicate traditional approaches towards authorship and persuasive engagement. Each of these works can also be considered what Espen Aarseth terms “ergodic literature” and “cybertext” (1), a work of calculated or computed expression which demands *non-trivial participation* by their readers in order to understand more fully the meaning(s) of that work.

While the term “cybertext” may have been abandoned by Aarseth during the last several years, it has nonetheless remained an integral concept for scholars of software, rhetoric, and literature to adequately describe the ways that textual processes function and how they make such functionality apparent to readers. Matthew G. Kirschenbaum has emphasized how Aarseth's focus on readers' “interaction[s] with a text's underlying formal processes” has set the stage for current critical inquiries into electronic literary studies (44). Collin Gifford Brooke observes that, among the theory's greatest strengths for rhetoricians, is that “it is the human-machine interaction [of a cybertext] that makes for virtualization,” which paves the way for practices of rhetorical invention (81). Stephen Ramsay highlights the problematic nature of critical interpretation (with an eye for literary criticism) when it comes to addressing the *potential* nature of cybertexts, since interpretive efforts inevitably reflect the realized experience of a linear expression of a given cybertext (41). N. Katherine Hayles has noted a weakness of Aarseth's cybertext theory in its “neglect” of social, political, and cultural concerns and contexts even as it attempts to address the nuances of literary potentiality (*My Mother* 37). Wardrip-Fruin argues against critical applications of Aarseth's theory, suggesting that few actually attempt to make use of it as argued, since “many of digital media's […] most important processes are not well described by the process of revealing” the mechanisms Aarseth outlines as central areas of inquiry for cybertext (161). However, Wardrip-Fruin acknowledges that such processes are still absolutely worthy of inquiry, and he points to software studies as a discipline that has successfully focused its efforts in this direction.

Despite the potential problems in using Aarseth's model of cybertext to refer to the hidden, obscured, or otherwise unfamiliar processes of digital texts, the term is nonetheless recognizable to many scholars interested in the critical study of software, code, and electronic literature. While it may not fully describe the procedural nature of a digital text, “cybertext,” with its built-in connotation of an *ergodic* quality, suggests a level of reader engagement that “hypertext” or “new media object” simply do not. Since this article focuses on an examination of the textual and procedural makeup of works that demonstrate (at least a significant component of) their mechanisms to their readers as a way to engage those readers with the texts, *cybertext* is considered an appropriate concept here to describe texts whose authors have consciously attempted to communicate meaning through their code structures and processes as well as through their expressive products (individual reading experiences).

The first text to be examined below is “Sarcophagus.txt,” a web page that makes use of Javascript in order to reconceptualize the acts of browser-based web reading and cybertextual composition. Through the act of reading the narratives generated by “Sarcophagus.txt,” shifting translations of code unfold vectors of meaning normally overlooked during the act of web browsing. “Sarcophagus.txt” continues a tradition of linguistic and procedural re-signification that asks audiences to actively reinterpret and contextualize an emerging argument presented in an unfamiliar and unconventional manner. The second is “PlaintextPerformance,” a partial record of the streams of data collected and intertwined during the course of a single performance event. The content of the text—the specific bits of data combined from separate sources—gives way to the emphasized temporality of the event. The reader is asked to make sense of the flow of information as a trajectory through time, space, and media rather than as a document or collection of documents made static through the replaying of the record. The third text to be examined is “forkbomb,” a single-line shell script designed to shut down a computer, whose generated human action replaces the visible factor of the code: once it is submitted for execution, there is literally no further response from the machine. Instead, the reaction is performed by the user(s) thereof, which could range from passive acceptance of the situation to totalitarian restriction of access to the system, forever limiting the potential action that could occur in relation thereto.

Rhetoric and the Study of Software

A key component of the rhetorical canons of *invention* and *memory* is the anticipation of the virtual—those probable (that is, pre-realized) constraints and preferences of an audience which influence a rhetor's construction of his or her argument for that audience. Similarly, considerations of *style*, *arrangement*, and *delivery* serve to illuminate how rhetors both enable and restrict the emergence of potentialities to suit their audience's reactions. A goal of rhetoric then is to understand exactly how language influences certain possibilities to become realized over others when used by a rhetor in a particular situation, and this goal is complicated when one considers how digital technologies enable and constrain those possibilities. Such complications are clarified somewhat when, following Brooke, we acknowledge that "[a] rhetoric of new media, rather than examining the *choices that have already been made* by writers, should prepare us as writers to *make our own choices*" (7). Indeed, Brooke calls for an "ecology of code" as part of this rhetoric of new media, "comprised of [...] all of those resources for the production of interfaces more broadly construed" from visual to spatial to code-based elements (48). By connecting together the classical canons and identified purposes of rhetoric with the potential of emerging interactive technologies, we are able to consider entirely new ways to communicate meaning to audiences and facilitate wide ranges of action as a result of our persuasive activities.

The power of language as a form of action enablement has been addressed by scholars such as Andy Clark, who notes that as language is learned, “cognitive shortcuts” which in turn facilitate the construction of mental relationships between, and ways of understanding, any number of concepts (78). Rhetoricians, as well as scholars interested in linguistic exchange and development, have the opportunity to connect together efforts of persuasion with the construction of novel shortcuts and associations between certain ideas, not unlike (to use Clark's example) the opening of pathways between neurons within the minds of specific audiences. This sort of relationship construction is already demonstrated in some rhetorical strategies, such as the use of metaphor, which involves an effort to persuade an audience to associate together two separate concepts or things as being more or less the same—at least for the purposes for which the rhetor is currently working.

This argument for language as means and form of rhetorical action may be easier to understand when pairing rhetoric with software studies, an emerging field in which scholars have, over the course of the last decade, undertaken serious efforts to explore the procedural logics of digital technologies and the code languages driving the use of those technologies. As Manovich observes in a critique leading to the development of software studies as a field, “[f]uture researchers will wonder why [contemporary] theoreticians, who had plenty of experience analyzing older cultural forms, did not try to described computer media's semiotic codes, modes of address, and audience reception patterns” (7). Some scholars, such as Hayles, have attempted to address this concern by probing the distinctions between “performative code” (the source code processed and executed by computer technology) and “figurative language” (semantic discourse used by humans to communicate with one another) (*My Mother* 127).

However, other critics, such as Florian Cramer, have noted the ultimate futility—or perhaps irrelevance—of such a distinction. For Cramer, “[t]here is nothing 'natural' about spoken language; it is a cultural construct and thus just as 'artificial' as any formal machine controls languages. To call programming languages 'machine languages' doesn't solve the problem either, as it obscures that 'machine languages' are human creations” (168). Wardrip-Fruin, who has distinguished code's functional *processes* from its *textual forms*, recognizes that “the processes of digital media operate both on and in terms of humanly meaningful elements and structures” (156). Others, especially those interested in critical examinations of code as meaningful texts, note that any distinctions like that made by Hayles, which separates symbolic value from machine function, “neglects the meaning that code bears for its human audiences” (Marino). The question remains unresolved, although it could serve as the foundation for further critical inquiry: what does one type of language do that the other does not, and what could the significance of this difference mean for critics interested in the rhetorical qualities of code-related communication?

Building upon Hayles' definitions, and keeping in mind the astute observations of critical code theorists, code differs from natural language beyond the former's aforementioned inability to be selectively interpreted (i.e. processed) by technologies in the more flexible ways that figurative language so often is by humans. First, code defines a range of potential action by way of its defined algorithmic process(es), with that action kept in stasis until its execution and expression. The uncompiled source code is always visibly only outside of its own execution—as Wendy Hui Kyong Chun has noted, source code exists functionally and symbolically as separate from compiled code programs (23). Second, code generates action despite its execution occurring beyond the perception of the audience; once a compiled program is called to perform processes, its code structure is obscured through a series of persuasive interfaces separating it from its audience. Teena A.M. Carnegie has pointed out the tendency for most of these interface "layers" to be obscured beneath the most "obvious," the user interface viewed through the screen (165). As a result, Carnegie suggests, rhetors and rhetoricians may overlook the possibilities for interaction between rhetor and audience that exist at and through each of these layers of interface (171). Scholars interested in technology studies or "digital rhetoric" are thus in an excellent position to scrutinize the role(s) of technology and code as a significant rhetorical component of textual creation and performance rather than simply a vehicle to transmit a text to an audience. This opportunity is especially significant given the increasing interest in experimentation (across multiple fields) with the dynamic possibilities of digital technologies for multimedia and multimodal approaches to writing.

The Oulipo

While digital technologies obviously facilitate the calculation of electronic texts, explorations of writing as a means of generating action through expression have their roots in the collected efforts of the Oulipo, a group of French artists, writers, and mathematicians in the mid-twentieth century. The Oulipo were interested in exploring the possibilities in literary forms and the boundaries of literary meaning-making when *medium* is championed rather than content, contrasting with the Romantic (or the more contemporary Beat) notion that the imagination let loose provides greater results than it could when constrained.

The members of the Oulipo were interested not so much in what a specific text looked like or said so much as *how* it was composed: what constraints were placed upon the author that enabled certain types of textual expressions. One well-known example of Oulipian composition is the “n+7” algorithm, in which an existing text has its nouns replaced with the seventh noun to follow the original term in a dictionary, resulting in a radically different textual *expression* with the same *structure* as the original text. Because the specific dictionary used by two different people for such inventive purposes could provide radically different results, the original text is never guaranteed to generate the same expression. This content, however, is subordinate to the process by which the text is “composed”—the language of the text, through its expression, gains focus as the significant action (the composition itself) rather than the meaning it transmits through that expression.

Marcel Bénabou, one of the Oulipo's founding members, suggests that, rather than make the process of invention an antagonist to the author's intended purpose or message, "linguistic constraints […] granted their arbitrary exigencies, directly create a sort of 'great vacuum' into which are sucked and retained whole quantities of elements which […] would otherwise remain concealed" (43). In other words, by emphasizing the algorithmic makeup of a text (or of a type of text), Oulipian writers make visible and accessible not just structure but also the rhetorical factors involved in the composition of that text. As David Rieder points out, *style*—along with the other canons of rhetoric—is a useful and appropriate concept from which to draw on knowledge that could illuminate the connections between literate and numerate approaches to writing; Rieder's example is the potential parallel that could be made between the rhetorical use of repetition and the computational use of looping functions within a software program. These connections are only some of the ways that language, in all its forms and structures, could help demonstrate and draw out a wider potential range of action than might otherwise have been apparent.

The most iconic work within the Oulipian repertoire may be Raymond Queneau's *Cent Mille Millardes de Poèmes* (“one hundred thousand billion poems”), a text comprised of 10 sonnets, each of which possesses the same rhyme scheme as the others. In its initial form, the sonnets were printed each on a separate sheet of paper with cuts made between every line, so that when assembled like a book, any line of any of the sonnets has the potential to be combined into a new poem with any line (from any of the original ten sonnets) that comes before or after it. As a result, the reader can construct any combination of one hundred trillion, or 10^{14}—that's 100,000,000,000,000—total potential combinations. Queneau refers to the “matrical analysis of language,” one way of describing the set of potential sonnets, as a means of pushing the recognized boundaries of potential literature beyond traditional, linear forms of composition (62). Considering the sonnets as a matrix also allows us to reflect on the poems’ visual structure; it is extremely difficult for a reader to disregard the process of navigating the individual components of the text—making fourteen decisions, each of which demands a reader choose from ten clearly separate options, in order to construct a specific poem. In terms of digital technologies, the reader would be said to engaging in the expression of an algorithm, realizing one outcome out of many possible outcomes within the algorithm's constraints.

Claude Berge describes one of the goals of the Oulipo as “the transposition of concepts existing in different branches of mathematics into the realm of words” (116). For the Oulipo, the possibilities that exist within formal constraints upon an author and audience, from manipulating rhyming patterns to exploring hypertextual non-linearity, offer individuals in either role to explore how those constraints help define invention, arrangement, and reading of a particular text.

Cybertextual Invention

Perhaps the best descriptor for computational Oulipian writing—both in the historical sense and in reference to the texts to be examined shortly—is *cybertext*, a term that refers in part to an ergodic work of art or literature, meaning the work requires a “non-trivial” engagement from a reader in order to be understood. Cybertexts also draw attention to their procedural mechanics, i.e. the means by which they function, as a significant component of reader engagement. According to Aarseth, a cybertext generates meaning from its “textons,” the stored and pre-processed texts that constitute the potential trajectories of a work, through the “involve[ment of] calculation in [its] production of scriptons” (75). Scriptons are “strings [of signs] as they appear to readers,” texts *presented to the reader* that have been expressed into their final form through a combined effort of the author and reader (Aarseth 62). In other words, the medium or media through which the text is expressed must make its mechanisms for scripton expression clear enough for the reader to engage with those mechanisms, which Aarseth refers to as “traversal functions” (62).

While the structure of a text is not inherently more important than its expressed content, for many cybertext authors—and the members of the Oulipo certainly fit this bill—it often becomes the locus of primary interest, both for the author anticipating potential readings of the text and the reader experiencing those readings. Brooke identifies Aarseth with such a stance, suggesting that the latter's approach “tends away from the referential, asking instead what practices are encouraged or enabled by a particular text, practices that include interpretation, but are not limited by it” (74). For Brooke and for other rhetoricians, this qualification is significant since the *potential* of a cybertext (those practices it enables) facilitates a wider range of approaches to rhetorical invention, including expressions formed it part by chance, than might be possible with a focus on interpreting particular and intentional scriptons.

> The use of chance (the relinquishment of authorial, or even reader-based, control over specific ways of reading a text) as a primary method of composing, especially for a text whose content emerges from the possibilities of its structure, has been critiqued by scholars of cybertext and digital media who—accurately—view the move away from conventional approaches to literary interpretation as emphasizing the potential for expression over what is being expressed. Hayles, for example, has argued that
>
> > [s]imply because cybertext theory predicts 576 different combinations [...] does not mean that all 576 combinations will be equally interesting or worthwhile. Nor does this number alone indicate the value of the theory, beyond setting up so many pigeonholes to be filled. Equally or more germane is what texts have done with the variables they choose to work with in exploring the nuances, complexities, and pleasures of a given configuration. (“What Cybertext”)

While it may be true that some readers are going to find certain readings more interesting than others, it is debatable whether or not some are more worthwhile. Each expression demonstrates the capabilities of applied constraints, and a reading of a constrained text incorporates a secondary system of constraint (the application of semantic interpretation and valuation) that allows the reader to categorize preferred or interesting expressions in contrast to uninteresting or non-preferred expressions. Aarseth characterizes the activity, and accordingly the value, of reading ergodic texts as “the dialectic between searching and finding” as a fundamental layer of human experience (91-92). Donald Knuth has pointed out the potential absurdity in approaching the study of processes via their individual tasks by suggesting that computer scientists “[m]ake a thorough analysis of everything your computer does during one second of computation,” which he estimates will include “several hundred thousand instructions” (4). The task is clearly and overwhelmingly difficult if not impossible, but the point is that efforts to seek out meaning in this fashion—to focus on the particulars rather than the potential—overlook the system and structure of processes that enables meaningful activity during that second.

What we will see in regards to the following cybertexts, then, is how the constraints upon reading and experiencing, as well as on combinatorially- or computationally-based invention, a given text (as both text and procedural system) further influence and generate action through that reading. Cybertexts created with digital technologies can make significant rhetorical use of the code underlying those technologies as either obvious or hidden components of the text. Also, both rhetor and reader are provided with an opportunity to engage critically with the fundamental qualities of technologies that may seem otherwise “transparent” in comparison to the content of a given text.

Sarcophagus.txt

"Sarcophagus.txt" by Titus Toledo is identified by the author as "an experiment in self-mutating automatic hyperfiction," a procedurally-generated narrative whose meaning is arguably clearer and made more significant in its Javascript code (an idea supported by the naming scheme for some of the Javascript files, including “deus.js”) than in any of its individual expressions. Fundamentally, "Sarcophagus.txt" can be viewed as a demonstration of writing whose persuasive qualities extend beyond the browser-rendered page. While it is only possible to view a particular narrative after the proper Javascript file has been initialized, it is the raw code of that same script which clarifies the scope of Toledo's efforts to create the *potential* for an overwhelming variety of hypertexts.

The composition of this text is achieved through its code “recipe,” a metaphor clear in the naming scheme of the code's functions, such as Soup(), taste(), and GenBoiledFowlRecipe(temperature), or even in the assignment of value to the variable Ingredients as

```
Ingredients = (Heat * Ingredients + Death) % 0x012261967
```

Toledo views the components of his narrative as ingredients that serve unique purposes; for example, the value of 'RawMeat' for one calculation reflects some

primarily human-oriented noun phrase, such as "tabloid press," guerrilla," or "crash test dummy," some of which are rendered in the HTML-page narrative as links to Google searches of those noun phrases, which expands the potential scope of the fiction far beyond “Sarcophagus.txt” itself and connects it to the larger web and all of each concept's overwhelmingly multiple presentations of potential meaning.

> Where Queneau offers 10^{14} potential sonnets, Toledo offers an even greater increase in the variety and number of tales his code can create, both in terms of length of a narrative and in the range of content for any given statement within that narrative. "Sarcophagus.txt" can include up to 82 sentences that can each include multiple variations on up to:
>
> 129 (mostly human-oriented) noun phrases (as noted above)
>
> 32 prepositions and phrases of relation (e.g. “informed by” or “in cahoots with”)
>
> 76 adverbs 189 adjectives
>
> 204 verbs (capable of multiple conjugations)
>
> 108 predicate phrases (e.g. “finally shows signs of X intelligent life”)
>
> 3 articles (e.g. a, an, the) 7 conjugations
>
> 72 general sentence structures (e.g. “The X and Y appear both Z.”)

Even though Toledo does not use the explicit syntax of the *array* as a data type to organize the values of each of the above categories, the logic of his code works the same as if he did—an embrace of data structuring that Manovich

refers to as “database logic that presents “a different model of what a world is like” (218-219). That is, an expression occurs when a particular array element number is called in order to make use of the text value associated with that element (i.e. identifying the 56th adjective). The modular nature of each statement, along with the number of potential values any component of the statement might possess, means that Toledo's fiction can produce texts whose meaning that varies wildly between expressions, even for the same reader. While it is true that the specific options available for any expression have been constrained by the choices Toledo opted to include, the number of those options mitigates “repetitive” expressions or readings by the same audience over time.

> While Queneau's sonnets are constrained by a very strict order—line 1 of sonnet A could never appear in the place of any sonnet's line 6—Toledo's generative script enables his sentences to be organized in almost any arrangement. As a cybertext, this fainter sense of constraint demands the reader draw connections between individual statements in ways that are simply unnecessary for Queneau (since the reader of a sonnet can rely on its metric structure and rhyme scheme to help form a framework for its understanding). Specifically, what is provided to the reader is a presentation of arrangement of scriptons whose relationship(s) with the other scriptons in the text is left to the reader's interpretation—a secondary and necessary act of invention in order for the text to “act.” As noted by Aarseth, “the computer as literary agent ultimately points beyond narrative and toward ergodic modes—dialogic forms of improvisation and free play” (62), which here refer to the types of *meaning-making* suggested through both the Javascript file and the narrative it expresses.

"Sarcophagus.txt" is thus an example of Oulipian writing even though the reader is removed from much of the compositional equation —since Javascript takes care of all of the variable computations, the reader is provided with one iteration of the text until he or she reloads the page in his or her browser so as to view another iteration thereof. There is no potential for subtler manipulations of individual variables to examine how conscious, purposeful changes to individual elements on the page might drastically change a reading of the text. While the mechanisms of the narrative are visible when viewing Toledo's Javascript file, they are unable to be directly (or at least not easily) manipulated by the reader. Instead, the reader is presented with the potential and the realized result but is locked out of the process of expressive realization. That said, this form of "locking-out" is different from that which occurs through the use of server-side code like PHP scripts, which Helen Burgess notes strips agency from both user and browser and places it in the relatively invisible web server, and its inaccessible "secret technology" (182). With client-side Javascript, the browser renders the code but it is accessible – the reader's engagement with the text is perhaps only fulfilled *when* he or she seeks out and examines the relevant script file(s).

“Sarcophagus.txt” manages to be successful in demanding work of its readers through the focus each work provides on the medium in which it exists. Because there are dozens of linked Google searches all throughout a given iteration of “Sarcophagus.txt,” provided in a randomized order, a reader is forced to compose his or her own set of connections between the cybertext and any results provided by Google. As Toledo, notes, the text “cannot [...] make sense, since it is not supposed to make sense, as no attempt is being made at sense or making sense, here or in the hereafter. If it makes sense, if it appears to make sense —sense being perceivable logic— it is only because the laws of chance has made it seemingly so.” As an experiment in form and function of web-friendly code, the text displaces the act of meaning-making *almost* entirely onto the reader, who is forced to discern significance out of a series of otherwise completely arbitrary hypertext links that happen to use the phrases contained in any of de Toledo's lists of variables.

The use of specific “calls” made by “Sarcophagus.txt” to the web at large, courtesy of Google's search algorithms, reflects the multiplicity of semantic values encoded into individual symbols and strings thereof. The Javascript variable values used in “Sarcophagus.txt” refer almost exclusively to in-page content (and a privileging of English language content, to boot) rather than to the structure of an HTML page itself, while much of the code is wrapped within the metaphorical frame of a cooking recipe, as if Toledo anticipates the reader feasting on the expressed results of the Javascript calculations.

In short, “Sarcophagus.txt” attempts to draw attention to the distinction between Hayles’ “performative code” and “figurative language” even as it suggests that the distinction can be blurred more easily than Hayles argues is possible (*My Mother* 127). “Sarcophagus.txt” performs the function of an Oulipian cybertext by providing a computationally-based method for a reader to engage with the *medium* of the HTML page and the wider web as constituted of more than HTML code itself: the possibilities of Javascript computation to create a dynamic reading experience that, statistically, will almost never be replicated perfectly between any two reading “sessions.” In addition, this text demonstrates the potential of a combinatorial type of writing that suggests the transmission of meaning across web pages based on the structural procedures of literary composition rather than focusing on the specific expression(s) of those procedures.

PlaintextPerformance

Where “Sarcophagus.txt” draws attention to the procedural structure of writing, “PlaintextPerformance” by Bjørn Magnhildøen highlights the inherent temporality of both composition and computation. The work serves as the archival remnant of a performance that demonstrated, and still demonstrates, the variety—and seeming incomprehensibility—of all the input and output that takes place concurrently, at different levels and different modes of communication, when an individual uses a computer (and when multiple computers interact via network). The result is an automatically-scrolling display of text that complicates 'reading' on the web through its temporally situated visualization. As the text continually forms itself, through its collection and translation of data into alphanumeric characters, the acts of scrolling (a reflection of the expressed representation of the assembled processes making up the text) and of translation from process data to expression into “plain text” become as important components of the text as the content visible on the screen at any given moment.

“PlaintextPerformance” may best be understood—in its existence as a performance—as a kind of *paratactical* expression of continually-processing computer activities, a text that draws from a grammar inclusive of all the routines running on the author's laptop (and, by extension, some of the routines running on others’ machines in proximity, as they communicate with the same wireless network as the author) without suggesting an inherent hierarchy stratifying those routines. (In contrast to *parataxis* is *hypotaxis*, an approach to narrative or speech wherein there is a subordinate relationship between the text's statements and ideas.) At any time, a personal computer is likely to have between 20 and 80 processes active, although the vast majority of these never become explicitly visible at the level of the “user interface” where they could be manipulated directly by the user.

This lack of process visibility is intentional: Hayles notes that, while code is often hidden or revealed strategically for aesthetic purposes (54), many programmers mean to obscure potential user-oriented code or system manipulation altogether from non-programmers (*My Mother* 130). While Hayles does not explore this detail further, a commonly-statedreason for such a restriction is to protect the user and his or her machine from damage (whether due to the user or to an outside party), such as if an individual were to begin deleting important system files or shutting down processes integral to the use of an operating system. Essentially, programmers attempt to save themselves frustration in the future by restricting users’ abilities to alter the design and coding work that the programmers have put into the initial creation of the software program. (While it is not provided as frequently as an argument against making software code available, such a user-based limitation also protects developers' proprietary code from competitors.)

To an extent, this can be seen as necessary: if bugs or security problems are discovered, patches (fixes and upgrades) to solve those problems, provided by the program's developer, can be compromised if every user has the potential to customize the code of his or her program. However, as a counter-point to this position, the free and libre open source software (FLOSS) movement argues that code should be made available so that a globally distributed body of would-be programmers can attempt to find and fix problems without waiting on the original developers to do so (“Open Source Initiative”).

Whatever the noted cause of obscurity, the significance of these hidden processes is that our control over what we read, write, and manipulate on a screen is determined to a great extent by constraints beyond the bounds of our general understanding and user-level 'logic,' with many of these imposed by programmers who provide little or no way for the user to interact with or modify them directly. In fact, as Joel Haefner has pointed out, the very logic of computer file systems and data organization is, to a great extent, a reflection of corporate hierarchy and value systems—something many non-programmers may not only find unfamiliar but altogether different from how they think and arrange information on our own (331-333). In other words, our actions, when incorporating digital technology into their executions, are not entirely intentional, and the implications of those actions very often extend beyond the range of our comprehension.

If we acknowledge the role of these unseen, and often distant, influences in the creation of what data we *see*, however, we can begin to recognize a sort of Oulipian method of construction for any digital text we interact with, as we work not just with semantic meaning of words and phrases within a given program but also with computational meaning in the interplay between processes being expressed during any given interval of time. By intertwining the streams of input and output from the author's laptop and nearby devices to mirror the continued communication occurring on each machine and over the network, Magnhildøen reworks a reading of situated computer use and comprehension. "PlaintextPerformance" makes plain(er) and explicit the role of that data into the body of the 'text' being performed, displaying how its continued intrusion into the on-screen data flow mirrors the temporal, continually moving and transforming nature of any text manipulated via computer that is normally considered to be a relatively 'static' and isolated document. The temporal constraints of computation are given priority over the preferred reading methods of the user so that the user can experience, in an unfamiliar way, the activities of his or her machine and those networked to it. In other words, the performance's expression of data collection and representation reflects the potential inherent in the actions available to and through digital technology. At the same time, "PlaintextPerformance" makes visible computer processes only as a byproduct of the recording of their expressions: as code executes, it also generates action through textual composition; it is a reminder that we interact as much with the products of code as much as with the code processes themselves.

It must be noted that “PlaintextPerformance,” like “Sarcophagus.txt,” functions in part thanks to the execution of Javascript. Unlike Toledo's approach, which connects together linked but separate Javascript and HTML files, Magnhildøen embeds his directly into the single HTML file that makes up his piece. Further, its utility is to create the scrolling effect that imitates the temporality of each process as it was executed and recorded; there are two set scrolling effects (one of which is commented out of the script's actual execution) complicated by randomized effects which might either speed up or momentarily reverse the scrolling display. The result is a demonstration of two key points: first, time always passes and thus more and more computational processes are executed; second, our belief that we possess the ability to exert some form of control over those continuing processes may likely be far stronger than the ability itself.

Magnhildøen describes the computer protocols on which his work is focused as “a lower level set of rules of the format of communication, and as statements reporting observations and experiences in the most fundamental terms without interpretation.” To do so, he relies on both the mathematical equations of computer processing, to construct his text and weave its parts together), and the audience, to construct a sense of understanding out of the viewing of each performance. In this way, Magnhildøen demonstrates and expands upon the potential, argued by Berge, in the capabilities of conceptual constraint and transposition: an author can provide his or her audience with a set of tools with which they might interpret not just text but also (Magnhildøen implies) to interpret interaction with a computer itself. With a stream of data that flows only momentarily into the field of vision on the screen, the device shows how information itself must be constrained and transposed into a type of image—words on the screen—so that we can cope with the recognition that *so much occurs* outside of our view and comprehension when we engage in the 'simple' acts of reading and writing.

Forkbomb

> Unlike the previous two texts, “Forkbomb,” a thirteen-character (space-inclusive) shell script by Denis Roio, who operates under the pseudonym 'Jaromil,' bears little resemblance to conventional language, since its characters are outside the bounds of any natural-language alphabet. “Forkbomb” is a concise piece of executable code that has arguably either one specific purpose (to shut down a computer) or an ambiguous multiplicity of purposes (to engage users of that computer in a set of responsive, individualized performances). In its entirety, the script (meant to be typed and executed in a command-line shell) reads:

```
:() {:|: &}; :
```

It may seem to be a piece of symbolic 'writing' without any audience other than the interpretive processes within a computer that will interpret and execute its commands. However, it is the lack of a visual component in the 'lifespan' of the code and its effects on involved *human,* as well as machine, audiences affected by its execution that make this composition, with all its relatively unfamiliar syntax and minimalistic logical style, so intriguing.

"Forkbomb" as a piece of code tells the computer to perform several tasks. First, it states that it will define a new function (identified by the colon and the pair of parentheses)—in simple terms, this is a definition of *something* that does *something*. Next, the script defines how that function will work; in this case, the computer is asked to call up two instances of the function at the same time (the pipe character | separates the two colons as instances of the command, but the computer will parse them both simultaneously), "in the background" rather than on the screen (a clarification indicated by the ampersand), every time the function is executed. Then, the definition is ended and the function is called for the first time. The result is that the code multiplies itself almost instantaneously within the computer's process tables, working not unlike a virus that infects and overloads brain cells. The computer is effectively gummed up, so to speak, with the cause being relatively undetectable once it is executed—once the "performative code" has been resolved, it is up to human behavior, a figurative and active language, to interpret the performance of the forkbomb. If, as Burgess suggests, server-side technologies shroud acts of networked computer use in secret and inaccessibility (182), then the forkbomb potentially redistributes that power of access back to the user, even if it is momentarily so.

The immediate response for many administrators is to reboot the “bombed” machine, shutting down the computer’s active processes and restarting “fresh” without the destructive script continuing to reconstitute itself and halting all other activities. While the reboot is the easiest response to perform in terms of immediate counter-actions, this act has several consequences: the loss of data in open files that had not been saved before the execution of the forkbomb, the interruption of work undertaken by and access for a number of users (depending upon the machine in question and whether it was a node of networked activity), and the (however temporary) re-opening of this security hole for future risk and a repetition of these actions. If the affected machine is a nexus for networked activity whose physical location is unavailable to most users, the bombing could prevent further activity on that machine until an administrator with access to the physical machine can successfully restart it.

Another response—for those who anticipate such scripts and destructive activity towards a machine—is to limit beforehand the number of simultaneous processes a single user can run at once. The forkbomb then becomes not so much a tool of vandalism, capable only within an unrestricted and thus 'unsafe' environment, as it is one of demonstrated security within a restricted setting, executed by network administrators to ensure that users are properly controlled. This second type of response reflects the values of social order and control possessed within a strongly hierarchical system: the available or allowed range of individual expression is explicitly measured, observed, and recorded. In such a setting, the failed execution of a forkbomb is a successful confirmation of administrative authority and power over the system’s user base; the script is transformed from a revolutionary act to a perfect demonstration of the inability for a user to profoundly affect the system.

Alexander R. Galloway and Eugene Thacker demonstrate the Perl-based code scripts for several “safe” forkbombs in their book *The Exploit*, noting that their forkbombs “autoprotect themselves from crashing; they strangle the machine but do not kill it [… through the display of ASCII art textures before quitting their execution.] By creating a high-stress environment within the computers processor, the artifacts of the machine itself become visible in the output” (176n17). The examples provided by Galloway and Thacker illustrate the self-administration prevalent with digital technologies: the citizen, the user, protects the systemic hierarchy of his or her machine by composing a “bomb” that threatens nothing and will *always* fizzle out. Yet, at the same time, an equal level of understanding is necessary to craft a script that will purposefully keep the user safe as it is to craft a script that will threaten the conventional stability of a computer system.

The “writing” involved in the performance of “Forkbomb” (not counting the ASCII output of the scripts provided by Galloway and Thacker) is “visual” in that it is almost entirely non-visual in its enactment: once the code is entered into the system through the shell, executed by the operating system, and subsequently replicated through an expanding set of background processes, it is inaccessible to author and audience alike. Its action is unnoticed directly, distinguishing the forkbomb’s expression from that of speech- (aurally noted) or typeface-based (visually noted) communication. However, despite this invisibility, the text’s mechanical structure and execution remains, to use Bolter and Grusin's term, a hypermediate quality of the text: even a user of the machine who may not know what is happening cannot ignore the event taking place or circumvent reacting to it. In other words, the situated, specifically computer-mediated experience of the forkbomb is never obscured from the act of reading or viewing it.

At the same time, the forkbomb is an applied example of the Oulipian poetry form called the *snowball*—a work whose every line increases or decreases in comparison to the previous line by one character, word, or other increment of measurement (for more on the varieties of snowballs generated by the Oulipo, see Rieder). In the case of “Forkbomb,” this linear increase is a doubling of the written content: one iteration of the function repeats itself as two, and each of those repeats as two, and so on. Where a print poem may end due to financial or practical printing constraints or inventive exhaustion, the forkbomb stops executing itself only when there is literally no more memory available within the computer system to allow the script to continue running. It is the medium, rather than the author or reader, which “ends” the expressive event. The variety of post-”bombing” reactions described earlier are the most interesting components of the performance, since they are the least predictable elements of the forkbomb, and they allow us to consider what we expect of readers' interactions with experiments in language and syntax.

The chain reaction caused by the forkbomb increases affected parties’ awareness of their momentary condition within the computer system, where they are disabled by their amputation from the computer with which they had been communicating. This affective consequence is made all the more significant given the nature of its cause: the interpretation of 13 non-alphabetic characters into a kind of multiplying virus within a computer processor, all of which takes place outside of the visible bounds of those audiences who respond and react in various ways to the outcome of the code's execution and expression. Since the composition and execution of “Forkbomb” can demonstrate itself as a validation of either the freedom or restricted ability of a system of computer users, the act of writing presents a clear capacity for and as action at multiple stages, including those of invention, expression, and interpretation.

Conclusions

The three digital media cybertexts described above serve as examples of the possibilities in combination and computation explored by groups like the Oulipo in order to understand the boundaries of language and literary composition. What may be most important in regards to these works is not just how they appear as "products" but how the semantic and computational codes that inform their structures provide for generative and mutating creations of meaning and interpretation. Further, an examination of these codes is *necessary* to understand how such meaning-making occurs, since these acts of making in turn generate and influence further action.

In this way, our understanding of writing can move beyond the word and the "surface" expressions of a text to incorporate the linguistic and computational processes that inform those particular expressions. Rather than looking through these mechanisms to specific content, we can examine these textual structures to comprehend more clearly how they influence the interpretations we draw from our methods of reading what might otherwise be understood as an inactive or "stand-alone" text. For rhetoricians in particular, such a modified approach could lead to a fuller understanding of how the rhetorical canons of invention, arrangement, and style are contingent upon the situated elements of audience, medium, and the ability of the former to modify and transform the latter in order to collaborate with the rhetor to create meaning through a text.

Works Cited

Aarseth, Espen. *Cybertext: Perspectives on Ergodic Literature*. Baltimore: Johns Hopkins UP, 1997. Print.

Aristotle. *On Rhetoric: A Theory of the Public Discourse*. 2nd ed. Trans. George A. Kennedy. Oxford: Oxford UP, 2007. Print.

Bénabou, Marcel. "Rule and Constraint." *Oulipo: A Primer for Potential Literature*. Ed. Warren Motte. Champaign, IL: Dalkey, 2007. 40-47. Print.

Berge, Claude. "Toward a Potential Analysis of Combinatorial Literature."

Oulipo: A Primer for Potential Literature. Ed. Warren Motte. Champaign, IL: Dalkey, 2007. 115-125. Print.

Bogost, Ian. *Persuasive Games: The Expressive Power of Videogames.* Cambridge, MA: MIT Press, 2007. Print.

Brooke, Collin Gifford. *Lingua Fracta: Towards a Rhetoric of New Media*. Cresskill, NJ: Hampton Press, 2009. Print.

Burgess, Helen J. “<?php>: 'Invisible' Code and the Mystique of Web Writing.” *From A to <A>: Keywords of Markup*. Eds. Bradley Dilger and Jeff Rice. Minneapolis: U of Minnesota Press, 2010. Print.

Burke, Kenneth. “(Nonsymbolic) Motion/(Symbolic) Action.” *Critical Inquiry* 4.4 (1978): 809-838. Web. 1 August 2011.

Carnegie, Teena A.M. “The Interface as Exordium: The Rhetoric of Interactivity.” *Computers and Composition* 26.3 (2009): 164-173. Web. 23 July 2011.

Chun, Wendy Hui Kyong. *Programmed Visions: Softwre and Memory*. Cambridge, MA: MIT Press, 2011. Print.

Clark, Andy. *Natural-Born Cyborgs: Minds, Technologies, and the Future of Human Intelligence*. Oxford and New York: Oxford UP, 2003. Print.

Cramer, Florian. “Language.” *Software Studies: A Lexicon*. Ed. Matthew Fuller. Cambridge, MA: MIT Press, 2008. 168-174. Print.

Fuller, Matthew. *Behind the Blip: Essays on the Culture of Software*. Brooklyn, NY: Autonomedia, 2003. Print.

—. “Introduction.” *Software Studies \ A Lexicon*. Ed. Matthew Fuller. Cambridge, MA: MIT Press. 1-13. Print.

—, ed. *Software Studies \ A Lexicon*. Cambridge, MA: MIT Press, 2008. Print.

Galloway, Alexander R. and Eugene Thacker. *The Exploit: A Theory of Networks.* Minneapolis: U of Minnesota Press, 2007. Print.

Halbritter, Bump. “Musical Rhetoric in Integrated Media Composition.” *Computers and Composition* 23.3 (2006): 317-334. Web. 2 February 2012.

Hayles, N. Katherine. *My Mother Was a Computer: Digital Subjects and Literary Texts*. Chicago: U of Chicago, 2005. Print.

—. “What Cybertext Theory Can't Do: A Riposte to Nick Montfort.” *Electronic*

Book Review (2001). Web. 1 August 2011.

Haefner, Joel. “The Politics of the Code.” *Computers and Composition* 16.3 (1999): 325-339. Web. 20 February 2011.

Kirschenbaum, Matthew G. *Mechanisms: New Media and the Forensic Imagination*. Cambridge, MA: MIT Press, 2008. Print.

Knuth, Donald. “Theory and Practice.” *Knuth: Preprints*. 28 August 1989. Web. 25 July 2012.

Kress, Gunther and Theo van Leeuwen. *Reading Images: The Grammar of Visual Design.* London: Routledge, 1996. Print.

Magnhildøen, Bjørn. “PlaintextPerformance: Author Description.” *Electronic Literature Collection.* Vol. 2. Feb. 2011. Web. 2 February 2012.

Manovich, Lev. *The Language of New Media*. Cambridge, MA: MIT Press, 2001. Print.

Marino, Mark C. “Critical Code Studies.” *Electronic Book Review.* 4 December 2006. Web. 10 November 2011.

Miller, Carolyn R. “Genre as Social Action.” *Quarterly Journal of Speech* 70 (1984): 151-167. Web. 25 June 2011.

Murray, Joddy. *Non-Discursive Rhetoric: Image and Affect in Multimodal Composition*. Albany: SUNY Press, 2009. Print.

Open Source Initiative. “Mission.” *Open Source Initiative*. Open Source Initiative. n.d. Web. 2 September 2011.

Queneau, Raymond. “Potential Literature.“ *Oulipo: A Primer for Potential Literature*. Ed. Warren Motte. Champaign, IL: Dalkey, 2007. 51-64. Print.

Ramsay, Stephen. *Reading Machines: Toward an Algorithmic Criticism.* Urbana: University of Illinois Press, 2011. Print.

Rieder, David. “Snowballs and Other Numerate Acts of Textuality: Exploring the ‘Alphanumeric’ Dimensions of (Visual) Rhetoric and Writing with ActionScript 3.” *Computers and Composition Online* (2010). Web. 24 June 2011.

Sondheim, Alan. “Introduction: Codework.” *American Book Review* 22.6 (2001): 1-2.

Toledo, Titus. *Sarcophagus.txt: An Experiment in Self-Mutating Automatic*

Hyperfiction. 2002. Web. 23 June 2011.

Wardrip-Fruin, Noah. *Expressive Processing: Digital Fictions, Computer Games, and Software Studies*. Cambridge, MA: MIT Press, 2009. Print.

Biography: Kevin Brock is a Ph.D. candidate in Communication, Rhetoric, and Digital Media at North Carolina State University. His work primarily focuses on the space shared by rhetoric and the critical studies of software and code, with a special interest in how code (as both practice and text) functions as a rhetorically powerful and significant form of contemporary communication. His dissertation in progress is titled "Engaging the Action-oriented Nature of Computation: Towards a Rhetorical Code Studies."

Technoculture Volume 2 (2012)

ISSN 1938-0526